链接: https://www.cnblogs.com/wupeiqi/p/9078770.html

<https://mp.weixin.qq.com/s?__biz=MzA5ODUzOTA0OQ==&mid=2651689264&idx=1&sn=11fbc3076ad08dc22bf80eb975c8ee14&chksm=8b693276bc1ebb60cdf08d14a5856050edee89406b9f94679c0b78765921cd04b3702cae6381&mpshare=1&scene=1&srcid=0601iRcbOSz3LQTWArSeIWqS#rd>

## 1、简述 OSI 七层协议。

OSI是一个开放性的通信系统互连参考模型，他是一个定义得非常好的协议规范。OSI模型有7层结构，每层都可以有几个子层。 OSI的7层从上到下分别是 7 应用层 6 表示层 5 会话层 4 传输层 3 网络层 2 数据链路层 1 物理层 ;其中高层(即7、6、5、4层)定义了应用程序的功能，下面3层(即3、2、1层)主要面向通过网络的端到端的数据流。

与其它计算机进行通讯的一个应用，它是对应应用程序的通信服务的。例如，一个没有通信功能的字处理程序就不能执行通信的代码，从事字处理工作的程序员也不关心OSI的第7层。但是，如果添加了一个传输文件的选项，那么字处理器的程序员就需要实现OSI的第7层。示例:TELNET，HTTP，FTP，NFS，SMTP等。

表示层

这一层的主要功能是定义数据格式及加密。例如，FTP允许你选择以二进制或ASCII格式传输。如果选择二进制，那么发送方和接收方不改变文件的内容。如果选择ASCII格式，发送方将把文本从发送方的字符集转换成标准的ASCII后发送数据。在接收方将标准的ASCII转换成接收方计算机的字符集。示例:加密，ASCII等。

会话层

它定义了如何开始、控制和结束一个会话，包括对多个双向消息的控制和管理，以便在只完成连续消息的一部分时可以通知应用，从而使表示层看到的数据是连续的，在某些情况下，如果表示层收到了所有的数据，则用数据代表表示层。示例:RPC，SQL等。

传输层

这层的功能包括是否选择差错恢复协议还是无差错恢复协议，及在同一主机上对不同应用的数据流的输入进行复用，还包括对收到的顺序不对的数据包的重新排序功能。示例:TCP，UDP，SPX。

网络层

这层对端到端的包传输进行定义，它定义了能够标识所有结点的逻辑地址，还定义了路由实现的方式和学习的方式。为了适应最大传输单元长度小于包长度的传输介质，网络层还定义了如何将一个包分解成更小的包的分段方法。示例:IP，IPX等。

数据链路层

它定义了在单个链路上如何传输数据。这些协议与被讨论的各种介质有关。示例:ATM，FDDI等。

物理层

OSI的物理层规范是有关传输介质的特这些规范通常也参考了其他组织制定的标准。连接头、帧、帧的使用、电流、编码及光调制等都属于各种物理层规范中的内容。物理层常用多个规范完成对所有细节的定义。示例:Rj45，802.3等。

## 2、什么是C/S和B/S架构？

1、C/S（Client/Server）结构，即大家熟知的客户机和服务器结构。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的通讯开销。目前大多数应用软件系

统都是Client/Server形式的两层结构，由于现在的软件应用系统正在向分布式的Web应用发展，Web和Client/Server应用都可以进行同样的业务处理，应用不同的模块共享逻辑组件；

2、B/S（Browser/Server）结构即浏览器和服务器结构。它是随着

Internet技术的兴起，对C/S结构的一种变化或者改进的结构。在这种结构下，用户工作界面是通过WWW浏览器来实现，极少部分事务逻辑在前端

（Browser）实现，但是主要事务逻辑在服务器端（Server）实现，形成所谓三层3-tier结构。这样就大大简化了客户端电脑载荷，减轻了系统维护与升级的成本和工作量，降低了用户的总体成本（TCO）。

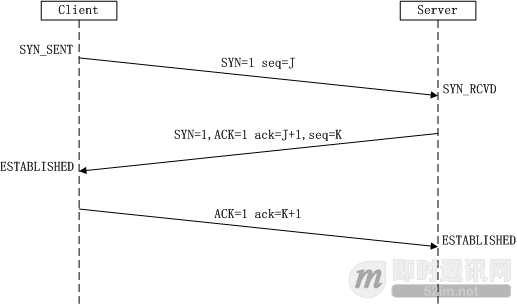
以目前的技术看，局域网建立B/S结构的网络应

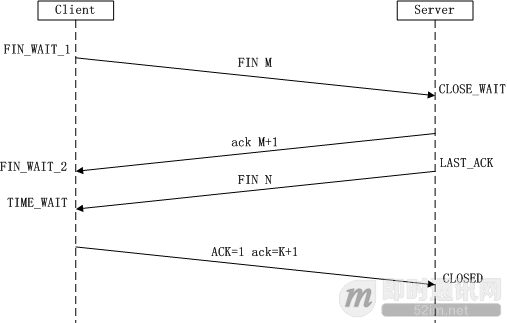
用，并通过Internet/Intranet模式下数据库应用，相对易于把握、成本也是较低的。它是一次性到位的开发，能实现不同的人员，从不同的地

点，以不同的接入方式（比如LAN,WAN,Internet/Intranet等）访问和操作共同的数据库；它能有效地保护数据平台和管理访问权限，服

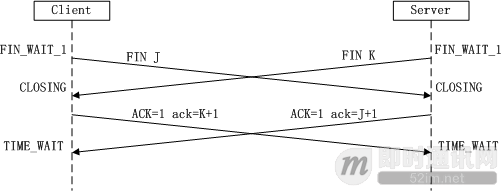
务器数据库也很安全。特别是在JAVA这样的跨平台语言出现之后，B/S架构管理软件更是方便、快捷、高效。

## 3、简述 三次握手、四次挥手的流程。

所谓三次握手（Three-Way Handshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。在socket编程中，这一过程由客户端执行connect来触发，整个流程如下图所示：  
   
  
**（1）第一次握手：**  
Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。  
**（2）第二次握手：**  
Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。  
**（3）第三次握手：**  
Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。  
**SYN攻击：**  
在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将产时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：

三次握手耳熟能详，四次挥手估计就少有人知道了。所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。在socket编程中，这一过程由客户端或服务端任一方执行close来触发，整个流程如下图所示：  
  
   
  
由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。

* 第一次挥手：  
  Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。
* 第二次挥手：  
  Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。
* 第三次挥手：  
  Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。
* 第四次挥手：  
  Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

上面是一方主动关闭，另一方被动关闭的情况，实际中还会出现同时发起主动关闭的情况，具体流程如下图：  
  
   
  
流程和状态在上图中已经很明了了，在此不再赘述，可以参考前面的四次挥手解析步骤。

## 4、什么是arp协议？

地址解析协议，即ARP（Address Resolution Protocol），是根据[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80)获取[物理地址](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%9C%B0%E5%9D%80/2129)的一个[TCP/IP协议](https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE)。[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA/455151)发送信息时将包含目标IP地址的ARP请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。地址解析协议是建立在网络中各个主机互相信任的基础上的，网络上的主机可以自主发送ARP应答消息，其他主机收到应答报文时不会检测该报文的真实性就会将其记入本机ARP缓存；由此攻击者就可以向某一主机发送伪ARP应答报文，使其发送的信息无法到达预期的主机或到达错误的主机，这就构成了一个[ARP欺骗](https://baike.baidu.com/item/ARP%E6%AC%BA%E9%AA%97)。[ARP命令](https://baike.baidu.com/item/ARP%E5%91%BD%E4%BB%A4)可用于查询本机ARP缓存中IP地址和[MAC地址](https://baike.baidu.com/item/MAC%E5%9C%B0%E5%9D%80)的对应关系、添加或删除静态对应关系等。相关协议有[RARP](https://baike.baidu.com/item/RARP)、[代理ARP](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86ARP)。[NDP](https://baike.baidu.com/item/NDP)用于在[IPv6](https://baike.baidu.com/item/IPv6)中代替地址解析协议。

## 5、TCP和UDP的区别？

TCP协议与UDP协议的区别

首先咱们弄清楚，TCP协议和UCP协议与TCP/IP协议的联系，很多人犯糊涂了，一直都是说TCP/IP协议与UDP协议的区别，我觉得这是没有从本质上弄清楚网络通信！

TCP/IP协议是一个协议簇。里面包括很多协议的。UDP只是其中的一个。之所以命名为TCP/IP协议，因为TCP,IP协议是两个很重要的协议，就用他两命名了。

TCP/IP协议集包括应用层,传输层，网络层，网络访问层。

其中应用层包括:

超文本传输协议(HTTP):万维网的基本协议.

文件传输(TFTP简单文件传输协议):

远程登录(Telnet),提供远程访问其它主机功能,它允许用户登录

internet主机,并在这台主机上执行命令.

网络管理(SNMP简单网络管理协议),该协议提供了监控网络设备的方法,以及配置管理,统计信息收集,性能管理及安全管理等.

域名系统(DNS),该系统用于在internet中将域名及其公共广播的网络节点转换成IP地址.

其次网络层包括:

Internet协议(IP)

Internet控制信息协议(ICMP)

地址解析协议(ARP)

反向地址解析协议(RARP)

最后说网络访问层:网络访问层又称作主机到网络层(host-to-network).网络访问层的功能包括IP地址与物理地址硬件的映射,以及将IP封装成帧.基于不同硬件类型的网络接口,网络访问层定义了和物理介质的连接.

当然我这里说得不够完善，TCP/IP协议本来就是一门学问，每一个分支都是一个很复杂的流程，但我相信每位学习软件开发的同学都有必要去仔细了解一番。

下面我着重讲解一下TCP协议和UDP协议的区别。

TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，只简单的描述下这三次对话的简单过程：主机A向主机B发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。

详细点说就是：（文章部分转载http://zhangjiangxing-gmail-com.iteye.com，主要是这个人讲解得很到位，的确很容易使人理解！）

TCP三次握手过程

1 主机A通过向主机B 发送一个含有同步序列号的标志位的数据段给主机B ,向主机B 请求建立连接,通过这个数据段,

主机A告诉主机B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.

2 主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机A,也告诉主机A两件事:

我已经收到你的请求了,你可以传输数据了;你要用哪佧序列号作为起始数据段来回应我

3 主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了

这样3次握手就完成了,主机A和主机B 就可以传输数据了.

3次握手的特点

没有应用层的数据

SYN这个标志位只有在TCP建产连接时才会被置1

握手完成后SYN标志位被置0

TCP建立连接要进行3次握手,而断开连接要进行4次

1 当主机A完成数据传输后,将控制位FIN置1,提出停止TCP连接的请求

2 主机B收到FIN后对其作出响应,确认这一方向上的TCP连接将关闭,将ACK置1

3 由B 端再提出反方向的关闭请求,将FIN置1

4 主机A对主机B的请求进行确认,将ACK置1,双方向的关闭结束.

由TCP的三次握手和四次断开可以看出,TCP使用面向连接的通信方式,大大提高了数据通信的可靠性,使发送数据端

和接收端在数据正式传输前就有了交互,为数据正式传输打下了可靠的基础

名词解释

ACK TCP报头的控制位之一,对数据进行确认.确认由目的端发出,用它来告诉发送端这个序列号之前的数据段

都收到了.比如,确认号为X,则表示前X-1个数据段都收到了,只有当ACK=1时,确认号才有效,当ACK=0时,确认号无效,这时会要求重传数据,保证数据的完整性.

SYN 同步序列号,TCP建立连接时将这个位置1

FIN 发送端完成发送任务位,当TCP完成数据传输需要断开时,提出断开连接的一方将这位置1

TCP的包头结构：

源端口 16位

目标端口 16位

序列号 32位

回应序号 32位

TCP头长度 4位

reserved 6位

控制代码 6位

窗口大小 16位

偏移量 16位

校验和 16位

选项 32位(可选)

这样我们得出了TCP包头的最小长度，为20字节。

UDP（User Data Protocol，用户数据报协议）

（1） UDP是一个非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

（2） 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

（3） UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小。

（4） 吞吐量不受拥挤控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。

（5）UDP使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。

（6）UDP是面向报文的。发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付给IP层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

我们经常使用“ping”命令来测试两台主机之间TCP/IP通信是否正常，其实“ping”命令的原理就是向对方主机发送UDP数据包，然后对方主机确认收到数据包，如果数据包是否到达的消息及时反馈回来，那么网络就是通的。

UDP的包头结构：

源端口 16位

目的端口 16位

长度 16位

校验和 16位

小结TCP与UDP的区别：

1.基于连接与无连接；

2.对系统资源的要求（TCP较多，UDP少）；

3.UDP程序结构较简单；

4.流模式与数据报模式 ；

5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

## 6、什么是局域网和广域网？

一、[局域网](http://www.so.com/s?q=%E5%B1%80%E5%9F%9F%E7%BD%91&ie=utf-8&src=internal_wenda_recommend_textn)   
局域网(Local Area Network)，简称LAN，是指在某一[区域](http://www.so.com/s?q=%E5%8C%BA%E5%9F%9F&ie=utf-8&src=internal_wenda_recommend_textn)内由多台[计算机](http://www.so.com/s?q=%E8%AE%A1%E7%AE%97%E6%9C%BA&ie=utf-8&src=internal_wenda_recommend_textn)互联成的计算机组。“某一区域”指的是同一[办公室](http://www.so.com/s?q=%E5%8A%9E%E5%85%AC%E5%AE%A4&ie=utf-8&src=internal_wenda_recommend_textn)、同一[建筑物](http://www.so.com/s?q=%E5%BB%BA%E7%AD%91%E7%89%A9&ie=utf-8&src=internal_wenda_recommend_textn)、同一公司和同一学校等，一般是方圆几千米以内。局域网可以实现文件管理、[应用软件](http://www.so.com/s?q=%E5%BA%94%E7%94%A8%E8%BD%AF%E4%BB%B6&ie=utf-8&src=internal_wenda_recommend_textn)共享、[打印机](http://www.so.com/s?q=%E6%89%93%E5%8D%B0%E6%9C%BA&ie=utf-8&src=internal_wenda_recommend_textn)共享、[扫描仪](http://www.so.com/s?q=%E6%89%AB%E6%8F%8F%E4%BB%AA&ie=utf-8&src=internal_wenda_recommend_textn)共享、工作组内的日程安排、[电子邮件](http://www.so.com/s?q=%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6&ie=utf-8&src=internal_wenda_recommend_textn)和[传真](http://www.so.com/s?q=%E4%BC%A0%E7%9C%9F&ie=utf-8&src=internal_wenda_recommend_textn)通信服务等功能。局域网是封闭型的，可以由办公室内的两台[计算机组成](http://www.so.com/s?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90&ie=utf-8&src=internal_wenda_recommend_textn)，也可以由一个公司内的上千台计算机组成。   
二、[广域网](http://www.so.com/s?q=%E5%B9%BF%E5%9F%9F%E7%BD%91&ie=utf-8&src=internal_wenda_recommend_textn)   
广域网(Wide Area Network)，简称WAN，是一种跨越大的、地域性的[计算机网络](http://www.so.com/s?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BD%91%E7%BB%9C&ie=utf-8&src=internal_wenda_recommend_textn)的集合。通常跨越省、市，甚至一个国家。广域网包括大大小小不同的[子网](http://www.so.com/s?q=%E5%AD%90%E7%BD%91&ie=utf-8&src=internal_wenda_recommend_textn)，子网可以是局域网，也可以是小型的广域网。

## 7、为何基于tcp协议的通信比基于udp协议的通信更可靠？

tcp:可靠 对方给了确认收到信息，才发下一个，如果没收到确认信息就重发

udp:不可靠 一直发数据，不需要对方回应

（1）UDP：user datagram protocol（用户数据报协议）

特点：

——1：面向无连接：传输数据之前源端和目的端不需要建立连接

——2：每个数据报的大小都限制在64k（8个字节）以内

——3：面向报文的不可靠协议（即发出去的数据不一定会接收到）

——4：传输速率快，效率高

——5：现实生活实例：邮局寄件，实时在线聊天，视频协议等等

（2）TCP：transmission control protocol（传输控制协议）

特点：

——1：面向连接：传输数据之前需要建立连接

——2：在连接过程中进行大量的数据传输

——3：通过“三次握手”的方式完成连接，是安全可靠的协议

——4：传输效率低，速度慢

## 8、什么是socket？简述基于tcp协议的套接字通信流程。

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，

Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，

对用户来说，一组简单的接口就是全部。

服务端：创建socket对象，绑定ip端口bind(), 设置最大链接数listen(), accept()与客户端

的connect()创建双向管道， send(), recv(),close()

客户端：创建socket对象，connect()与服务端accept()创建双向管道 , send(), recv(),close()

## 9、什么是粘包？ socket 中造成粘包的原因是什么？ 哪些情况会发生粘包现象？

如客户端只recv(1024), 可结果比1024长那怎么办，只好在服务器端的IO缓冲区里把客户端还没收走的暂时存下来，等客户端下次再来收，所以当客户端第2次调用recv(1024)就会首先把上次没收完的数据先收下来，再收df命令的结果。

这个现象叫做粘包，就是指两次结果粘到一起了。它的发生主要是因为socket缓冲区导致的。

所谓粘包问题原因：（1）主要还是因为接收方不知道消息之间的界限，不知道一次性提取多少字节的数据所造成的。（2）发送方引起的粘包是由TCP协议本身造成的，TCP为提高传输效率，发送方往往要收集到足够多的数据后才发送一个TCP段。若连续几次需要send的数据都很少，通常TCP会根据优化算法把这些数据合成一个TCP段后一次发送出去，这样接收方就收到了粘包数据。

发生粘包的情况：（1）发送端需要等缓冲区满才发送出去，造成粘包（发送数据时间间隔很短，数据了很小，会合到一起，产生粘包）（2）接收方不及时接收缓冲区的包，造成多个包接收（客户端发送了一段数据，服务端只收了一小部分，服务端下次再收的时候还是从缓冲区拿上次遗留的数据，产生粘包）

## 10、IO多路复用的作用？

把socket交给操作系统去监控，相当于**找个代理人(select), 去收快递。快递到了,就通知用户，用户自己去取。**

**阻塞I/O只能阻塞一个I/O操作，而I/O复用模型能够阻塞多个I/O操作，所以才叫做多路复用**

## 11、什么是防火墙以及作用？

## 12、select、poll、epoll 模型的区别？



## 13、简述 进程、线程、协程的区别 以及应用场景？

1.线程和进程：

线程是属于进程的，线程运行在进程空间内，同一进程所产生的线程共享同一内存空间，当进程退出时该进程所产生的线程都会被强制退出并清除。线程可与属于同一进程的其它线程共享进程所拥有的全部资源，但是其本身基本上不拥有系统资源，只拥有一点在运行中必不可少的信息(如程序计数器、一组寄存器和栈)。

2.线程、进程与协程：

线程和进程的操作是由程序触发系统接口，最后的执行者是系统；协程的操作则是程序员

协程存在的意义：对于多线程应用，CPU通过切片的方式来切换线程间的执行，线程切换时需要耗时（保持状态，下次继续）。协程，则只使用一个线程，在一个线程中规定某个代码块执行顺序。

协程的适用场景： 当程序中存在大量不需要CPU的操作时（IO），适用于协程；

## 14、GIL锁是什么鬼？

Gil全局解释锁延伸扩展

作用 : 保证同一时刻只有一个线程能使用到cpu

解释 : 当我们使用多线程的时候,在一个进程中只有一个GIL锁,那么这多个线程中谁拿到GIL谁就可以使用cpu(ps:多个进程有多个Gil锁)

问题1: 什么时候会释放Gil锁,

答:  1.遇到像 i/o操作这种 会有时间空闲情况 造成cpu闲置的情况会释放Gil

      2会有一个专门ticks进行计数 一旦ticks数值达到100这个时候释放Gil锁 线程之间开始竞争Gil锁(说明:ticks这个数值可以进行设置来延长或者缩减获得Gil锁的线程使用cpu的时间)

问题2: 互斥锁和Gil锁的关系

Gil锁  :保证同一时刻只有一个线程能使用到cpu

互斥锁 : 多线程时,保证修改共享数据时有序的修改,不会产生数据修改混乱

首先假设只有一个进程,这个进程中有两个线程 Thread1,Thread2, 要修改共享的数据date, 并且有互斥锁

执行以下步骤

(1)多线程运行，假设Thread1获得GIL可以使用cpu，这时Thread1获得 互斥锁lock,Thread1可以改date数据(但并没有开始修改数据)

(2)Thread1线程在修改date数据前发生了 i/o操作 或者 ticks计数满100

(注意就是没有运行到修改data数据),这个时候 Thread1 让出了Gil,Gil锁可以被竞争

(3) Thread1 和 Thread2 开始竞争Gil (注意:如果Thread1是因为i/o 阻塞 让出的Gil Thread2必定拿到Gil,如果Thread1是因为ticks计数满100让出Gil这个时候Thread1 和 Thread2 公平竞争)

(4)假设 Thread2正好获得了GIL, 运行代码去修改共享数据date,由于Thread1有互斥锁lock，所以Thread2无法更改共享数据date,这时Thread2让出Gil锁, GIL锁再次发生竞争

(5)假设Thread1又抢到GIL，由于其有互斥锁Lock所以其可以继续修改共享数据data,当Thread1修改完数据释放互斥锁lock,Thread2在获得GIL与lock后才可对data进行修改

以上描述了 互斥锁和Gil锁的 一个关系

总结:

1.线程锁是fine-grained(细粒度)的锁，程序员需要自行加/解锁来保证线程安全;

2.全局解释锁是coarse-grained(粗粒度)的锁，语言层面本身维护着一个全局的锁机制用来保证线程安全;

前一种方式比较典型的是 Java, Jython 等, 后一种方式比较典型的是 CPython (即Python)。

所以，线程互斥锁和全局解释锁的关系，前一个是程序自定义的线程锁，后一个是CPython强加给解释器的全局线程锁，从本质上来讲，两者并没有太大区别，都是为了线程安全而设计的，只是GIL比较恶心，没法移除，并强行执行，导致python的多线程成为伪并发的多线程，每一时刻，python一个进程下有且只有一个线程能获取到GIL锁执行任务。

## 15、Python中如何使用线程池和进程池？

最近准备爬取某电商网站的数据，先不考虑代理、分布式，先说效率问题（当然你要是请求的太快就会被封掉，亲测，400个请求过去，服务器直接拒绝连接，心碎），步入正题。一般情况下小白的我们第一个想到的是for循环，这个可是单线程啊。那我们考虑for循环直接开他个5个线程，问题来了，如果有一个url请求还没有回来，后面的就干等，这么用多线程等于没用，到处贴创可贴。

**二、性能考虑**

　　确定要用多线程或者多进程了，那我们到底是用多线程还是多进程，有些人对多进程和多线程有一定的偏见，就因为python的GIL锁，下面我们说一下这两个东西的差别。

**三、多线程：**

 　　一般情况下我们启动一个.py文件，就等于启动了一个进程，一个进程里面默认有一个线程工作，我们使用的多线程的意思就是在一个进程里面启用多个线程。但问题来了，为什么要使用多线程呢？我知道启动一个进程的时候需要创建一些内存空间，就相当于一间房子，我们要在这个房子里面干活，你可以想一个人就等于一个线程，你房子里面有10个人的空间跟有20个人的空间，正常情况下是不一样的，因为我们知道线程和线程之间默认是可以通信的（进程之间默认是不可以通信的，不过可以用技术实现，比如说管道）。可以多线程为了保证计算数据的正确性，所以出现了GIL锁，保证同一时间只能有一个线程在计算。GIL锁你可以基本理解为，比如在这个房间里要算一笔账，在同一时间内只能有一个人在算这笔账，想一个问题，如果这笔账5个人就能算清楚，我需要10平米的房间就行，那为什么要请10个人，花20平米呢？所以并不是开的线程越多越好。但是，但是，但是，注意大家不用动脑筋（CPU计算）算这笔账的时候可以去干别的事（比如说5个人分工，各算一部分），比如说各自把自己算完后的结果记录在账本上以便后面对账，这个的话每个人都有自己的账本，所以多线程适合IO操作，记住了就算是适合IO操作，也不代表说人越多越好，所以这个量还是得根据实际情况而定。

　　示例：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 线程池

**四、多进程：**

 　　上面我们介绍了多线程（线程池），现在我们聊聊进程池，我们知道一个进程占用一个CPU，现在的配置CPU一般都是4核，我们启动两个进程就是分别在两个CPU里面（两个内核）各运行一个进程，我知道进程里面才有线程，默认是一个。但是有个缺点，按照上面的说法，开两个进程占用的内存空间是开一个进程占用内存空间的2倍。CPU就占用了2个核，电脑还得干别的事儿对吧，不能冒冒失失瞎用。开的太多是不是其他程序就得等着，我们思考一下，占用这么多的内存空间，利用了多个CPU的优点为了什么？CPU是用来做什么的？没错就是用来计算的，所以在CPU密集运算的情况下建议用多进程。注意，具体要开几个进程，根据机器的实际配置和实际生产情况而定。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 进程池

**总结：**

 　　1、多线程适合IO密集型程序

　　 2、多进程适合CPU密集运算型程序

**五、协程：**

　　协程：又称微线程纤程。英文名Coroutine。那协程到底是个什么东西，通俗的讲就是比线程还要小的线程，所以才叫微线程。

　　主要作用：有人要问了，在python中线程是原子操作（意思就是说一句话或者一个动作就能搞定的操作或者计算），怎么还有个叫协程的呢？

　　优点：

　　　　1、使用高并发、高扩展、低性能的；一个CPU支持上万的协程都不是问题。所以很适合用于高并发处理。

　　　　2、无需线程的上下文切换开销（乍一看，什么意思呢？我们都知道python实际上是就是单线程，那都是怎么实现高并发操作呢，就是CPU高速的切换，每个任务都干一点，最后看上去是一起完事儿的，肉眼感觉就是多线程、多进程）

　　缺点：

　　　　1、无法利用CPU的多核优点，这个好理解，进程里面包含线程，而协程就是细分后的线程，也就是说一个进程里面首先是线程其后才是协程，那肯定是用不了多核了，不过可以多进程配合，使用CPU的密集运算，平时我们用不到。

　　1、一般情况下用的比较多的是asyncio或者是gevent这两个技术实现协程，asyncio是python自带的技术，gevent第三方库，个人比较喜欢gevent这个技术。

**gevent：**

　　　　安装：gevent需要安装greenlet，因为它是使用到了greenlet这个库。

|  |  |
| --- | --- |
| 1  2 | pip3 install greenlet  pip3 install gevent |

　　　　1、gevent的基本实现，按照下面的写法，程序启动后将会开启许许多多的协程，反而特别影响性能。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif gevent+requests

　　　　2、有一个改进版本，就是可以设置到底让它一次发起多少个请求（被忘了，协程=高并发现实之一）。其实里面就是利用gevnet下的pool模块里面的Pool控制每次请求的数量。

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif gevent+reqeust+Pool（控制每次请求数量）

　　　　3、还有一版本，每次我们都要装greenlet和gevent这肯定是没法子，但是，我们上面写的这个改进版还是有点麻烦，所以就有人写了100多行代码把它们给搞到了一起，对就是搞到了一起，叫grequests，就是前者两个技术的结合。

|  |  |
| --- | --- |
|  |  |

　　　　这个版本是不是特别变态，直接把requests、greenlet、gevent、Pool都省的导入了，但是装还是要装的，有人说从下面代码中我没看到Pool的参数啊，grequests.map(request\_list,size=5)，size就是你要同时开几个协程，还有参数你得点进去看，是不是很牛，很轻松

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif grequests

 　　　　结果返回一个列表，你可以再迭代一下就行了。

## 16、threading.local的作用？

有个概念叫做线程局部变量。一般我们对多线程中的全局变量都会加锁处理，这样的变量是共享变量，每一个线程都能够读写变量，为了保持同步我们会做枷锁处理。可是有些变量初始化以后。我们仅仅想让他们在每一个线程中一直存在。相当于一个线程内的共享变量，线程之间又是隔离的。python threading模块中就提供了这么一个类，叫做local。

## 17、进程之间如何进行通信？

**1、无名管道通信**

　　无名管道（ pipe ）：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

**2、高级管道通信**

　　高级管道（popen）：将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前程序的子进程，这种方式我们成为高级管道方式。

**3、有名管道通信**

　　有名管道 （named pipe） ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

**4、消息队列通信**

　　消息队列（ message queue ） ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

**5、信号量通信**

　　信号量（ semophore ） ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

**6、信号**

　　信号 （ sinal ） ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

**7、共享内存通信**

　　共享内存（ shared memory ） ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

**8、套接字通信**

　　套接字（ socket ） ： 套接口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

## 18、什么是并发和并行？

实际开发中很多人会因为并发，并行，串行，同步，异步等名词搞混，搞不清楚这些名称的意思和之间对应的关系。其实这也是实际开发中必须掌握一些知识点。掌握和理解这些概念也有助于我们去更好去开好。

1.并发编程又叫多线程编程。

在程序中，往往有很多很耗时的工作，比如上传文件、下载文件、跟客户聊天需要长时间建立连接。这种时候，一个线程是服务不了多个用户的，会产生因为资源独占产生的等待问题。并发的实质是一个物理CPU(也可以多个物理CPU) 在若干道程序之间多路复用，并发性是对有限物理资源强制行使多用户共享以提高效率。（买票问题并发进行）

并发当有多个线程在操作时,如果系统只有一个CPU,则它根本不可能真正同时进行一个以上的线程，它只能把CPU运行时间划分成若干个时间段,再将时间 段分配给各个线程执行，在一个时间段的线程代码运行时，其它线程处于挂起状。.这种方式我们称之为并发(Concurrent)。

2.“并行”指两个或两个以上事件或活动在同一时刻发生。在多道程序环境下，并行性使多个程序同一时刻可在不同CPU上同时执行。（hadoop集群就是并行计算的）

当系统有一个以上CPU时,则线程的操作有可能非并发。当一个CPU执行一个线程时，另一个CPU可以执行另一个线程，两个线程互不抢占CPU资源，可以同时进行，这种方式我们称之为并行(Parallel)。

并发和并行

并发和并行是即相似又有区别的两个概念，并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔内发生。在多道程序环境下，并发性是指在一段时间内宏观上有多个程序在同时运行，但在单处理机系统中，每一时刻却仅能有一道程序执行，故微观上这些程序只能是分时地交替执行。倘若在计算机系统中有多个处理机，则这些可以并发执行的程序便可被分配到多个处理机上，实现并行执行，即利用每个处理机来处理一个可并发执行的程序，这样，多个程序便可以同时执行。

3.串行、并行：

并行和串行指的是任务的执行方式。串行是指多个任务时，各个任务按顺序执行，完成一个之后才能进行下一个。并行指的是多个任务可以同时执行，异步是多个任务并行的前提条件。

4.同步、异步：

指的是能否开启新的线程。同步不能开启新的线程，异步可以。

异步：异步和同步是相对的，同步就是顺序执行，执行完一个再执行下一个，需要等待、协调运行。异步就是彼此独立,在等待某事件的过程中继续做自己的事，不需要等待这一事件完成后再工作。线程就是实现异步的一个方式。异步是让调用方法的主线程不需要同步等待另一线程的完成，从而可以让主线程干其它的事情。

异步和多线程并不是一个同等关系,异步是最终目的,多线程只是我们实现异步的一种手段。异步是当一个调用请求发送给被调用者,而调用者不用等待其结果的返回而可以做其它的事情。实现异步可以采用多线程技术或则交给另外的进程来处理。

5.多线程：多线程是程序设计的逻辑层概念，它是进程中并发运行的一段代码。多线程可以实现线程间的切换执行。

## 19、进程锁和线程锁的作用？

在分布式集群系统的开发中，线程锁往往并不能支持全部场景的使用，必须引入新的技术方案分布式锁。

　　线程锁:大家都不陌生，主要用来给方法、代码块加锁。当某个方法或者代码块使用锁时，那么在同一时刻至多仅有有一个线程在执行该段代码。当有多个线程访问同一对象的加锁方法/代码块时，同一时间只有一个线程在执行，其余线程必须要等待当前线程执行完之后才能执行该代码段。但是，其余线程是可以访问该对象中的非加锁代码块的。

　　进程锁:也是为了控制同一操作系统中多个进程访问一个共享资源，只是因为程序的独立性，各个进程是无法控制其他进程对资源的访问的，但是可以使用本地系统的信号量控制（操作系统基本知识）。

　　分布式锁:当多个进程不在同一个系统之中时，使用分布式锁控制多个进程对资源的访问。

## 20、解释什么是异步非阻塞？

**1、前言**

　　前一段时间出去面试，被问到同步、异步与阻塞、非阻塞的区别。我一时半会没有想出来，作为一个工作三年的人来说，实在很惭愧。我当时理解同步、异步属于两个进程中间的协作关系，例如使用浏览器访问一个网站，需要多次请求服务端，才能加载完整个页面的内容。同步的操作如下：浏览器首先发送第一个请求，等待服务器回复后，再发送第二个请求，依次类推，直到所有请求完成。异步的操作如下：浏览器发送第一个请求，可以不用等待服务器返回，可以继续发送第二个请求。阻塞与非阻塞属于进程的API执行动作的方式，例如进行需要read数据，阻塞方式操作流程是：如果没有数据，则read会一直等着数据到来，才能进行后续的动作；而非阻塞则是read没有到数据后，则可以进行后续的动作，当有数据的时候再回来读取。通常linux网络API默认都是阻塞的，例如connect、send、recv等。回答后感觉自己心里没有底，底层的关系到底是什么样的，比较虚，没能深入理解。

**2、深入理解分析**

　　回来以后，赶紧上网好好查查，加深学习一下。这两个概念在工作中经常用到这些，例如在linux网络IO中涉及到如下模型：

（1）阻塞式IO

（2）非阻塞式IO

（3）IO多路复用

（4）信号驱动IO

（5）异步IO

在知乎上面看到一些解释如下：

截图如下：

## 21、路由器和交换机的区别？

路由器、交换机

· 交换机(Switch)是一种基于MAC（网卡的硬件地址）识别，能完成封装转发数据包功能的网络设备。交换机可以“学习”MAC地址，并把其存放在内部地址表中，通过在数据帧的始发者和目标接收者之间建立临时的交换路径，使数据帧直接由源地址到达目的地址。现在的交换机分为：二层交换机，三层交换机或是更高层的交换机。三层交换机同样可以有路由的功能，而且比低端路由器的转发速率更快。它的主要特点是：一次路由，多次转发。

·路由器(Router)亦称选径器，是在网络层实现互连的设备。它比网桥更加复杂，也具有更大的灵活性。路由器有更强的异种网互连能力，连接对象包括局域网和广域网。过去路由器多用于广域网，近年来，由于路由器性能有了很大提高，价格下降到与网桥接近，因此在局域网互连中也越来越多地使用路由器。路由器是一种连接多个网络或网段的网络设备，它能将不同网络或网段之间的数据信息进行“翻译”，以使它们能够相互“读”懂对方的数据，从而构成一个更大的网络。路由器有两大典型功能，即数据通道功能和控制功能。数据通道功能包括转发决定、背板转发以及输出链路调度等，一般由特定的硬件来完成；控制功能一般用软件来实现，包括与相邻路由器之间的信息交换、系统配置、系统管理等。

## 22、什么是域名解析？

域名解析英文名：DNS (domain name resolution)

域名注册好之后，只说明你对这个域名拥有了使用权，在进行域名解析之后，这个域名才能发挥它的作用，经过解析的域名可以作为网址访问自己的网站，也可以作为电子邮箱的后缀，因此“域名解析”是使用域名的必备环节。

注册域名之后如何才能看到自己的网站内容，用一个专业术语就叫“域名解析”。

人们都习惯记忆域名，但机器间互相只认 IP 地址，域名与 IP 地址之间是一 一对应的，它们之间的转换工作称为域名解析，域名解析需要由专门的域名解析服务器来完成，解析过程是自动进行的。

示例：

如您在访问一个网站时，会输入网址 www.net.cn，之后即会显示对应的网站页面，这个情景在后台是什么样的呢？网站的内容所在服务器只有 IP 地址，将域名与 IP 地址绑定（即域名解析），引导访问者获取网站服务器中的网页内容，即可实现输入网址进行网站访问。

如您在使用电子邮箱时，需要在互联网上表明您邮箱地址与邮箱服务器的关系，通过域名解析之后您和对方，才能够正确的发送邮件，邮件才能够正确的找到您的邮箱。

## 23、如何修改本地hosts文件？

【第一步】

hosts文件位置：例如，Windows系统安装在C:\WINDOWS，那么Hosts文件就在C:\WINDOWS\system32\drivers\etc中。

【第二步】

hosts文件打开与编辑：hosts文件是一个纯文本的文件，用普通的文本编辑软件如记事本等都能打开和编辑。

【第三步】

绑定服务器IP测试：新起一行添加网站地址与服务器的IP地址，例如：

192.168.1.1 www.abc.com

复制代码

确认后保存、退出。（注意：没有#号）

【第四步】

验证绑定后：通过Ping命令，探测网站解析情况， 当ping返回的结果是之前绑定的网站IP地址，说明绑定成功。

c:\>ping www.abc.com

正在 Ping www.abc.com [192.168.1.1] 具有 32 字节的数据:

来自 192.168.1.1 的回复: 字节=32 时间=43ms TTL=253

来自 192.168.1.1 的回复: 字节=32 时间=37ms TTL=253

复制代码

【第五步】

检测网站状态：在浏览器输入网站的地址，如 www.abc.com，如果网站可以正常打开，说明服务器工作状态正常。如果不能正常打开，说明服务器状态异常。

Windows, IP地址, 记事本, 服务器, 如何, IP地址, IP地址

## 24、生产者消费者模型应用场景及优势？

## 25、什么是cdn？

## 26、LVS是什么及作用？

## 27、Nginx是什么及作用？

负载平衡

负载均衡是由多台服务器以对称的方式组成一个服务器集合，每台服务器都具有等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。

通过某种负载分担技术，将外部发送来的请求均匀分配到对称结构中的某一台服务器上，而接收到请求的服务器独立地回应客户的请求。

均衡负载能够平均分配客户请求到服务器列阵，籍此提供快速获取重要数据，解决大量并发访问服务问题。

这种群集技术可以用最少的投资获得接近于大型主机的性能。

**正向代理和反向代理**

从用途上来讲：

正向代理的典型用途为了防火墙内的局域网客户端提供访问Internet的途经。

正向代理还可以使用缓存特性减少网络使用率。

反向代理的典型用途是将防火墙后面的服务器提供给Internet用户访问。

反向代理服务器还可以为后端的多台服务器提供负载均衡，

或者后端较慢的服务器提供缓冲服务。

从安全性来讲：

正向代理允许客户通过它访问任意网站并且隐藏客户端自身，因此你必须采取安全措施以确保仅为经过授权服务器提供服务。

反向代理对外是透明的，访问者并不知道自己访问的是一个代理。

## 28、keepalived是什么及作用?

## 29、haproxy是什么以及作用？

## 30、什么是负载均衡？

## 31、什么是rpc及应用场景？

## 32、简述 asynio模块的作用和应用场景。

## 33、简述 gevent模块的作用和应用场景。

## 34、twisted框架的使用和应用？

## 35、HTTP请求行、请求头、请求体详解

学习Web开发不好好学习HTTP报文，将会“打拳不练功，到老一场空”，你花在犯迷糊上的时间比你沉下心来学习HTTP的时间肯定会多很多。  
**HTTP请求报文解剖**   
HTTP请求报文由3部分组成（**请求行+请求头+请求体**）：   
  
   
下面是一个实际的请求报文：   
  
①是请求方法，GET和POST是最常见的HTTP方法，除此以外还包括DELETE、HEAD、OPTIONS、PUT、TRACE。不过，当前的大多数浏览器只支持GET和POST，Spring 3.0提供了一个HiddenHttpMethodFilter，允许你通过“\_method”的表单参数指定这些特殊的HTTP方法（实际上还是通过POST提交表单）。服务端配置了HiddenHttpMethodFilter后，Spring会根据\_method参数指定的值模拟出相应的HTTP方法，这样，就可以使用这些HTTP方法对处理方法进行映射了。   
  
②为请求对应的URL地址，它和报文头的Host属性组成完整的请求URL，③是协议名称及版本号。   
  
④是HTTP的报文头，报文头包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息。   
  
⑤是报文体，它将一个页面表单中的组件值通过param1=value1&param2=value2的键值对形式编码成一个格式化串，它承载多个请求参数的数据。不但报文体可以传递请求参数，请求URL也可以通过类似于“/chapter15/user.html? param1=value1&param2=value2”的方式传递请求参数。   
对照上面的请求报文，我们把它进一步分解，你可以看到一幅更详细的结构图：   


## 36、HTTP相应HEADER

**HTTP响应报文解剖**   
  
**响应报文结构**   
  
HTTP的响应报文也由三部分组成（**响应行+响应头+响应体**）：   
  
   
  
以下是一个实际的HTTP响应报文：   
  
   
  
①报文协议及版本；   
②状态码及状态描述；   
③响应报文头，也是由多个属性组成；   
④响应报文体，即我们真正要的“干货”。   
  
**响应状态码**   
  
和请求报文相比，响应报文多了一个“响应状态码”，它以“清晰明确”的语言告诉客户端本次请求的处理结果。   
  
HTTP的响应状态码由5段组成： 

* 1xx 消息，一般是告诉客户端，请求已经收到了，正在处理，别急...
* 2xx 处理成功，一般表示：请求收悉、我明白你要的、请求已受理、已经处理完成等信息.
* 3xx 重定向到其它地方。它让客户端再发起一个请求以完成整个处理。
* 4xx 处理发生错误，责任在客户端，如客户端的请求一个不存在的资源，客户端未被授权，禁止访问等。
* 5xx 处理发生错误，责任在服务端，如服务端抛出异常，路由出错，HTTP版本不支持等。

以下是几个常见的状态码：   
  
**200 OK**   
  
你最希望看到的，即处理成功！   
  
**303 See Other**   
  
我把你redirect到其它的页面，目标的URL通过响应报文头的Location告诉你。

**引用**

悟空：师傅给个桃吧，走了一天了http://www.iteye.com/images/smiles/icon_confused.gif   
唐僧：我哪有桃啊！去王母娘娘那找吧http://www.iteye.com/images/smiles/icon_wink.gif

**304 Not Modified**   
  
告诉客户端，你请求的这个资源至你上次取得后，并没有更改，你直接用你本地的缓存吧，我很忙哦，你能不能少来烦我啊！   
  
**404 Not Found**   
  
你最不希望看到的，即找不到页面。如你在google上找到一个页面，点击这个链接返回404，表示这个页面已经被网站删除了，google那边的记录只是美好的回忆。   
  
**500 Internal Server Error**   
  
看到这个错误，你就应该查查服务端的日志了，肯定抛出了一堆异常，别睡了，起来改BUG去吧！   
  
  
其它的状态码参见：<http://en.wikipedia.org/wiki/List_of_HTTP_status_codes>   
  
  
有些响应码，Web应用服务器会自动给生成。你可以通过HttpServletResponse的API设置状态码： 

**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. //设置状态码，状态码在HttpServletResponse中通过一系列的常量预定义了，如SC\_ACCEPTED，SC\_OK
2. **void**    setStatus(**int** sc)

**常见的HTTP响应报文头属性**   
  
**Cache-Control**   
  
响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。   
  
下面，的设置让客户端对响应内容缓存3600秒，也即在3600秒内，如果客户再次访问该资源，直接从客户端的缓存中返回内容给客户，不要再从服务端获取（当然，这个功能是靠客户端实现的，服务端只是通过这个属性提示客户端“应该这么做”，做不做，还是决定于客户端，如果是自己宣称支持HTTP的客户端，则就应该这样实现）。 

**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. Cache-Control: max-age=3600

**ETag**   
  
一个代表响应服务端资源（如页面）版本的报文头属性，如果某个服务端资源发生变化了，这个ETag就会相应发生变化。它是Cache-Control的有益补充，可以让客户端“更智能”地处理什么时候要从服务端取资源，什么时候可以直接从缓存中返回响应。   
  
关于ETag的说明，你可以参见：<http://en.wikipedia.org/wiki/HTTP_ETag>。   
Spring 3.0还专门为此提供了一个org.springframework.web.filter.ShallowEtagHeaderFilter（实现原理很简单，对JSP输出的内容MD5，这样内容有变化ETag就相应变化了），用于生成响应的ETag，因为这东东确实可以帮助减少请求和响应的交互。   
  
下面是一个ETag：

**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. ETag: "737060cd8c284d8af7ad3082f209582d"

**Location**   
  
我们在JSP中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过响应报文头的Location属性告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中： 

**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. Location: http://www.iteye.com

**Set-Cookie**   
  
服务端可以设置客户端的Cookie，其原理就是通过这个响应报文头属性实现的： 

**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1

**其它HTTP响应报文头属性**   
  
更多其它的HTTP响应头报文，参见：<http://en.wikipedia.org/wiki/List_of_HTTP_header_fields>   
**如何写HTTP请求报文头**   
在服务端可以通过HttpServletResponse的API写响应报文头的属性： 

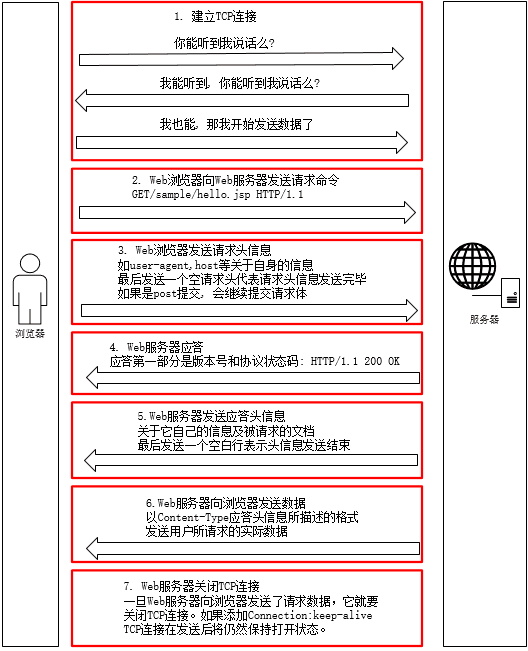
**Java代码  [收藏代码](https://blog.csdn.net/u010256388/article/details/68491509)**

1. //添加一个响应报文头属性
2. **void**    setHeader(String name, String value)

## 37、完整的HTTP请求

网址：<https://blog.csdn.net/yezitoo/article/details/78193794>

**一、 HTTP请求和响应步骤**



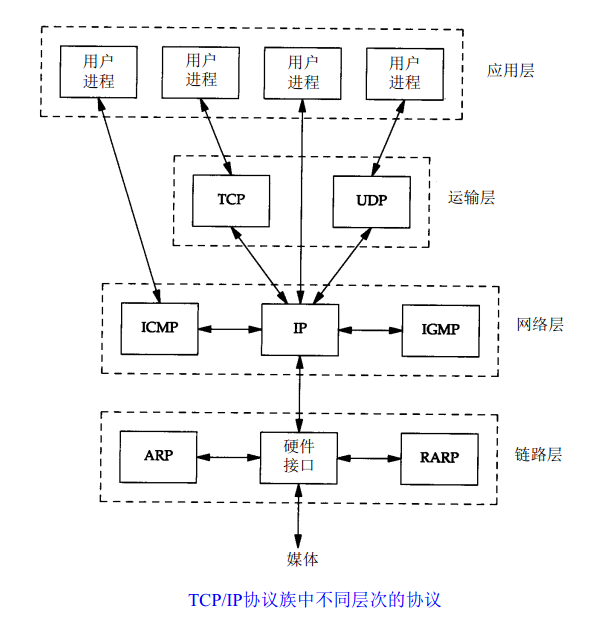
图片来自：[理解Http请求与响应](http://android.jobbole.com/85218/)

以上完整表示了HTTP请求和响应的7个步骤，下面从TCP/IP协议模型的角度来理解HTTP请求和响应如何传递的。

**二、TCP/IP协议**

TCP/IP协议模型（Transmission Control Protocol/Internet Protocol），包含了一系列构成互联网基础的网络协议，是Internet的核心协议，通过20多年的发展已日渐成熟，并被广泛应用于局域网和广域网中，目前已成为事实上的国际标准。TCP/IP协议簇是一组不同层次上的多个协议的组合，通常被认为是一个四层协议系统，与OSI的七层模型相对应。

HTTP协议就是基于TCP/IP协议模型来传输信息的。



(1). 链路层

也称作数据链路层或网络接口层（在第一个图中为网络接口层和硬件层），通常包括操作系统中的设备驱动程序和计算机中对应的网络接口卡。它们一起处理与电缆（或其他任何传输媒介）的物理接口细节。ARP（地址解析协议）和RARP（逆地址解析协议）是某些网络接口（如以太网和令牌环网）使用的特殊协议，用来转换IP层和网络接口层使用的地址。

(2). 网络层

也称作互联网层（在第一个图中为网际层），处理分组在网络中的活动，例如分组的选路。在TCP/IP协议族中，网络层协议包括IP协议（网际协议），ICMP协议（Internet互联网控制报文协议），以及IGMP协议（Internet组管理协议）。

IP是一种网络层协议，提供的是一种不可靠的服务，它只是尽可能快地把分组从源结点送到目的结点，但是并不提供任何可靠性保证。同时被TCP和UDP使用。TCP和UDP的每组数据都通过端系统和每个中间路由器中的IP层在互联网中进行传输。

ICMP是IP协议的附属协议。IP层用它来与其他主机或路由器交换错误报文和其他重要信息。

IGMP是Internet组管理协议。它用来把一个UDP数据报多播到多个主机。

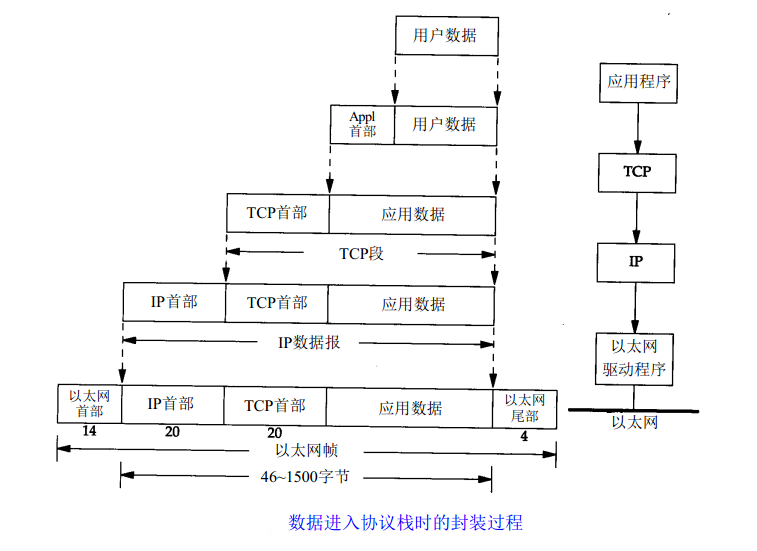
(3). 传输层

主要为两台主机上的应用程序提供端到端的通信。在TCP/IP协议族中，有两个互不相同的传输协议：TCP（传输控制协议）和UDP（用户数据报协议）。

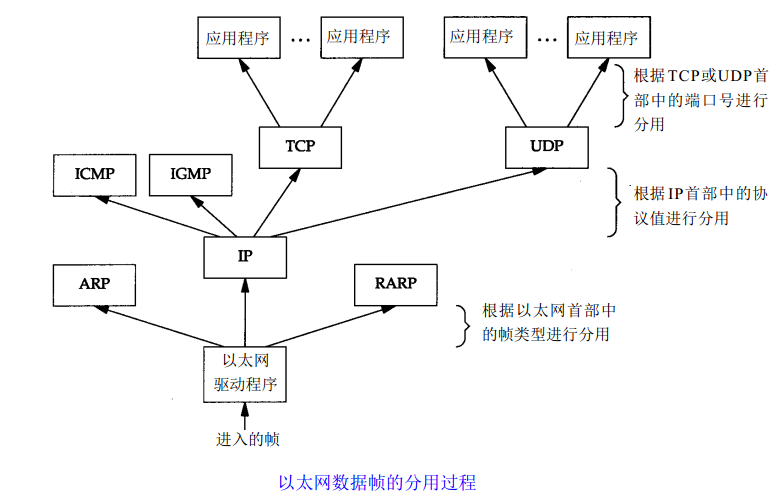
TCP为两台主机提供高可靠性的数据通信。它所做的工作包括把应用程序交给它的数据分成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于运输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。为了提供可靠的服务，TCP采用了超时重传、发送和接收端到端的确认分组等机制。

UDP则为应用层提供一种非常简单的服务。它只是把称作数据报的分组从一台主机发送到另一台主机，但并不保证该数据报能到达另一端。一个数据报是指从发送方传输到接收方的一个信息单元（例如，发送方指定的一定字节数的信息）。UDP协议任何必需的可靠性必须由应用层来提供。  
(4). 应用层

应用层决定了向用户提供应用服务时通信的活动。TCP/IP 协议族内预存了各类通用的应用服务。包括 HTTP，FTP（File Transfer Protocol，文件传输协议），DNS（Domain Name System，域名系统）服务。



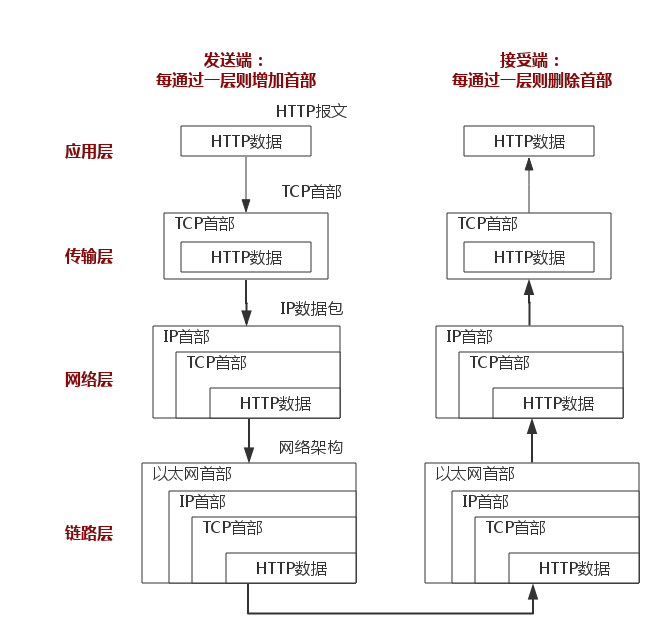
当应用程序用TCP传送数据时，数据被送入协议栈中，然后逐个通过每一层直到被当作一串比特流送入网络。其中每一层对收到的数据都要增加一些首部信息（有时还要增加尾部信息），该过程如图所示。



当目的主机收到一个以太网数据帧时，数据就开始从协议栈中由底向上升，同时去掉各层协议加上的报文首部。每层协议盒都要去检查报文首部中的协议标识，以确定接收数据的上层协议。这个过程称作分用（Demultiplexing）。协议是通过目的端口号、源I P地址和源端口号进行解包的。

通过以上步骤我们从TCP/IP模型的角度来理解了一次HTTP请求与响应的过程。

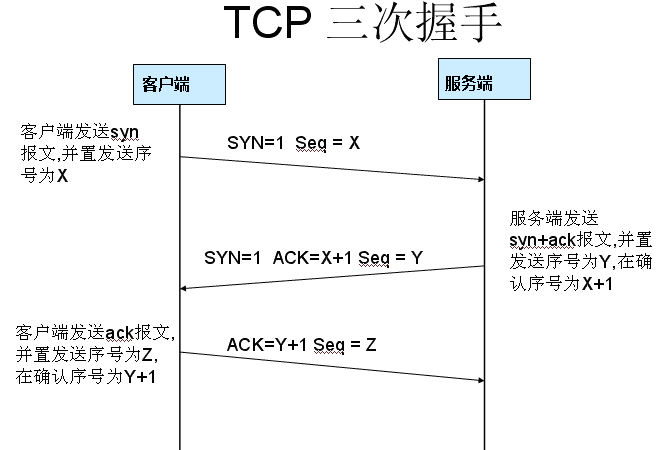
下面这张图更清楚明白：



下面具体来看如何进行一步步操作的。

**三、TCP三次握手**

TCP是面向连接的，无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接。在TCP/IP协议中，TCP协议提供可靠的连接服务，连接是通过三次握手进行初始化的。三次握手的目的是同步连接双方的序列号和确认号并交换 TCP窗口大小信息。



第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN\_SEND状态，等待服务器的确认；

第二次握手：服务器收到SYN报文段。服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置Acknowledgment Number为x+1(Sequence Number+1)；同时，自己自己还要发送SYN请求信息，将SYN位置为1，Sequence Number为y；服务器端将上述所有信息放到一个报文段（即SYN+ACK报文段）中，一并发送给客户端，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK报文段。然后将Acknowledgment Number设置为y+1，向服务器发送ACK报文段，这个报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。

**为什么要三次握手**

为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

**具体例子：**“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。”

**四、HTTP协议**

**Http是什么？**

通俗来讲，他就是计算机通过网络进行通信的规则，是一个基于请求与响应，无状态的，应用层的协议，常基于TCP/IP协议传输数据。目前任何终端（手机，笔记本电脑。。）之间进行任何一种通信都必须按照Http协议进行，否则无法连接。

**四个基于：**

**请求与响应：**客户端发送请求，服务器端响应数据

**无状态的：**协议对于事务处理没有记忆能力，客户端第一次与服务器建立连接发送请求时需要进行一系列的安全认证匹配等，因此增加页面等待时间，当客户端向服务器端发送请求，服务器端响应完毕后，两者断开连接，也不保存连接状态，一刀两断！恩断义绝！从此路人！下一次客户端向同样的服务器发送请求时，由于他们之前已经遗忘了彼此，所以需要重新建立连接。

**应用层：**Http是属于应用层的协议，配合TCP/IP使用。

**TCP/IP：**Http使用TCP作为它的支撑运输协议。HTTP客户机发起一个与服务器的TCP连接，一旦连接建立，浏览器（客户机）和服务器进程就可以通过套接字接口访问TCP。

**针对无状态的一些解决策略：**

有时需要对用户之前的HTTP通信状态进行保存，比如执行一次登陆操作，在30分钟内所有的请求都不需要再次登陆。于是引入了Cookie技术。

HTTP/1.1想出了持久连接（HTTP keep-alive）方法。其特点是，只要任意一端没有明确提出断开连接，则保持TCP连接状态，在请求首部字段中的Connection: keep-alive即为表明使用了持久连接。  
等等还有很多。。。。。。

下面开始讲解重头戏：HTTP请求报文，响应报文，对应于上述步骤的2，3，4，5，6。

HTTP报文是面向文本的，报文中的每一个字段都是一些ASCII码串，各个字段的长度是不确定的。HTTP有两类报文：请求报文和响应报文。

**五、HTTP请求报文**

一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成，下图给出了请求报文的一般格式。



**1.请求行**

请求行分为三个部分：请求方法、请求地址和协议版本

**请求方法**

HTTP/1.1 定义的请求方法有8种：GET、POST、PUT、DELETE、PATCH、HEAD、OPTIONS、TRACE。

最常的两种GET和POST，如果是RESTful接口的话一般会用到GET、POST、DELETE、PUT。

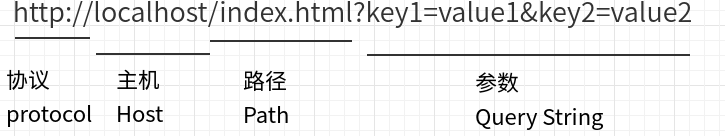
**请求地址**

URL:统一资源定位符，是一种自愿位置的抽象唯一识别方法。

组成：<协议>：//<主机>：<端口>/<路径>

**端口和路径有时可以省略（HTTP默认端口号是80）**

如下例：



有时会带参数，GET请求

**协议版本**

协议版本的格式为：HTTP/主版本号.次版本号，常用的有HTTP/1.0和HTTP/1.1

**2.请求头部**

请求头部为请求报文添加了一些附加信息，由“名/值”对组成，每行一对，名和值之间使用冒号分隔。

常见请求头如下：



请求头部的最后会有一个空行，表示请求头部结束，接下来为请求数据，这一行非常重要，必不可少。

**3.请求数据**

可选部分，比如GET请求就没有请求数据。

下面是一个POST方法的请求报文：

POST 　/index.php　HTTP/1.1 　　 请求行  
Host: localhost  
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2　　请求头  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
Accept-Language: zh-cn,zh;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: <http://localhost/>  
Content-Length：25  
Content-Type：application/x-www-form-urlencoded  
　　空行  
username=aa&password=1234　　请求数据

**六、HTTP响应报文**



HTTP响应报文主要由状态行、响应头部、空行以及响应数据组成。

**1.状态行**

由3部分组成，分别为：协议版本，状态码，状态码描述。

其中协议版本与请求报文一致，状态码描述是对状态码的简单描述，所以这里就只介绍状态码。

**状态码**

状态代码为3位数字。  
1xx：指示信息--表示请求已接收，继续处理。  
2xx：成功--表示请求已被成功接收、理解、接受。  
3xx：重定向--要完成请求必须进行更进一步的操作。  
4xx：客户端错误--请求有语法错误或请求无法实现。  
5xx：服务器端错误--服务器未能实现合法的请求。

下面列举几个常见的：



**2.响应头部**

与请求头部类似，为响应报文添加了一些附加信息

常见响应头部如下：



**3.响应数据**

用于存放需要返回给客户端的数据信息。

下面是一个响应报文的实例：

HTTP/1.1 200 OK　　状态行  
Date: Sun, 17 Mar 2013 08:12:54 GMT　　响应头部  
Server: Apache/2.2.8 (Win32) PHP/5.2.5  
X-Powered-By: PHP/5.2.5  
Set-Cookie: PHPSESSID=c0huq7pdkmm5gg6osoe3mgjmm3; path=/  
Expires: Thu, 19 Nov 1981 08:52:00 GMT  
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0  
Pragma: no-cache  
Content-Length: 4393  
Keep-Alive: timeout=5, max=100  
Connection: Keep-Alive  
Content-Type: text/html; charset=utf-8  
　　空行

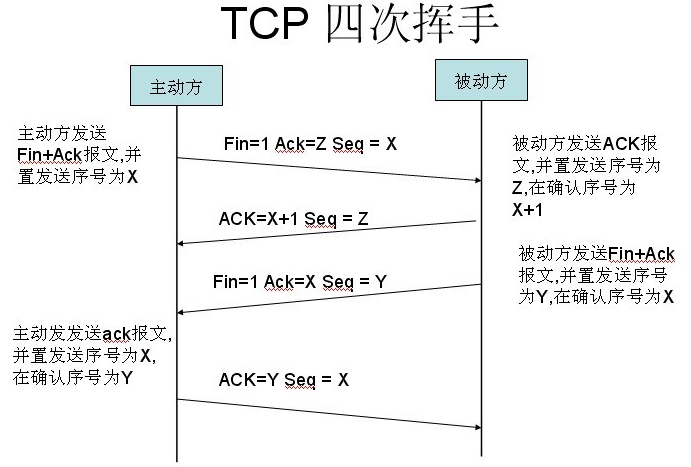
<html>　　响应数据  
<head>  
<title>HTTP响应示例<title>  
</head>  
<body>  
Hello HTTP!  
</body>  
</html>

关于请求头部和响应头部的知识点很多，这里只是简单介绍。

通过以上步骤，数据已经传递完毕，HTTP/1.1会维持持久连接，但持续一段时间总会有关闭连接的时候，这时候据需要断开TCP连接。

**七、TCP四次挥手**

当客户端和服务器通过三次握手建立了TCP连接以后，当数据传送完毕，肯定是要断开TCP连接的啊。那对于TCP的断开连接，这里就有了神秘的“四次分手”。



第一次分手：主机1（可以使客户端，也可以是服务器端），设置Sequence Number，向主机2发送一个FIN报文段；此时，主机1进入FIN\_WAIT\_1状态；这表示主机1没有数据要发送给主机2了；

第二次分手：主机2收到了主机1发送的FIN报文段，向主机1回一个ACK报文段，Acknowledgment Number为Sequence Number加1；主机1进入FIN\_WAIT\_2状态；主机2告诉主机1，我“同意”你的关闭请求；

第三次分手：主机2向主机1发送FIN报文段，请求关闭连接，同时主机2进入LAST\_ACK状态；

第四次分手：主机1收到主机2发送的FIN报文段，向主机2发送ACK报文段，然后主机1进入TIME\_WAIT状态；主机2收到主机1的ACK报文段以后，就关闭连接；此时，主机1等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机1也可以关闭连接了。

**为什么要四次分手**

TCP协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。TCP是全双工模式，这就意味着，当主机1发出FIN报文段时，只是表示主机1已经没有数据要发送了，主机1告诉主机2，它的数据已经全部发送完毕了；但是，这个时候主机1还是可以接受来自主机2的数据；当主机2返回ACK报文段时，表示它已经知道主机1没有数据发送了，但是主机2还是可以发送数据到主机1的；当主机2也发送了FIN报文段时，这个时候就表示主机2也没有数据要发送了，就会告诉主机1，我也没有数据要发送了，之后彼此就会愉快的中断这次TCP连接。

通过以上步骤便完成了HTTP的请求和响应，进行了数据传递，这其中涉及到需要知识点，都进行了逐一了解。

## 38、HTTP响应头和请求头信息对照表

HTTP请求头提供了关于请求，响应或者其他的发送实体的信息。HTTP的头信息包括通用头、请求头、响应头和实体头四个部分。每个头域由一个域名，冒号（:）和域值三部分组成。

* **通用头标**：即可用于请求，也可用于响应，是作为一个整体而不是特定资源与事务相关联。
* **请求头标**：允许客户端传递关于自身的信息和希望的响应形式。
* **响应头标**：服务器和于传递自身信息的响应。
* **实体头标**：定义被传送资源的信息。即可用于请求，也可用于响应。

根据以上分类的HTTP请求头介绍可以[参考此文](http://www.jb51.net/article/51951.htm)，本工具根据请求和输出分为Request和Response两部分。

### 38.1 HTTP Request Header 请求头

| **Header** | **解释** | **示例** |
| --- | --- | --- |
| Accept | 指定客户端能够接收的内容类型 | Accept: text/plain, text/html |
| Accept-Charset | 浏览器可以接受的字符编码集。 | Accept-Charset: iso-8859-5 |
| Accept-Encoding | 指定浏览器可以支持的web服务器返回内容压缩编码类型。 | Accept-Encoding: compress, gzip |
| Accept-Language | 浏览器可接受的语言 | Accept-Language: en,zh |
| Accept-Ranges | 可以请求网页实体的一个或者多个子范围字段 | Accept-Ranges: bytes |
| Authorization | HTTP授权的授权证书 | Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Cache-Control | 指定请求和响应遵循的缓存机制 | Cache-Control: no-cache |
| Connection | 表示是否需要持久连接。（HTTP 1.1默认进行持久连接） | Connection: close |
| Cookie | HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器。 | Cookie: $Version=1; Skin=new; |
| Content-Length | 请求的内容长度 | Content-Length: 348 |
| Content-Type | 请求的与实体对应的MIME信息 | Content-Type: application/x-www-form-urlencoded |
| Date | 请求发送的日期和时间 | Date: Tue, 15 Nov 2010 08:12:31 GMT |
| Expect | 请求的特定的服务器行为 | Expect: 100-continue |
| From | 发出请求的用户的Email | From: user@email.com |
| Host | 指定请求的服务器的域名和端口号 | Host: www.zcmhi.com |
| If-Match | 只有请求内容与实体相匹配才有效 | If-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Modified-Since | 如果请求的部分在指定时间之后被修改则请求成功，未被修改则返回304代码 | If-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| If-None-Match | 如果内容未改变返回304代码，参数为服务器先前发送的Etag，与服务器回应的Etag比较判断是否改变 | If-None-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Range | 如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。参数也为Etag | If-Range: “737060cd8c284d8af7ad3082f209582d” |
| If-Unmodified-Since | 只在实体在指定时间之后未被修改才请求成功 | If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| Max-Forwards | 限制信息通过代理和网关传送的时间 | Max-Forwards: 10 |
| Pragma | 用来包含实现特定的指令 | Pragma: no-cache |
| Proxy-Authorization | 连接到代理的授权证书 | Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Range | 只请求实体的一部分，指定范围 | Range: bytes=500-999 |
| Referer | 先前网页的地址，当前请求网页紧随其后,即来路 | Referer: http://www.zcmhi.com/archives/71.html |
| TE | 客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息 | TE: trailers,deflate;q=0.5 |
| Upgrade | 向服务器指定某种传输协议以便服务器进行转换（如果支持） | Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11 |
| User-Agent | User-Agent的内容包含发出请求的用户信息 | User-Agent: Mozilla/5.0 (Linux; X11) |
| Via | 通知中间网关或代理服务器地址，通信协议 | Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1) |
| Warning | 关于消息实体的警告信息 | Warn: 199 Miscellaneous warning |

### 38.2 HTTP Responses Header 响应头

| **Header** | **解释** | **示例** |
| --- | --- | --- |
| Accept-Ranges | 表明服务器是否支持指定范围请求及哪种类型的分段请求 | Accept-Ranges: bytes |
| Age | 从原始服务器到代理缓存形成的估算时间（以秒计，非负） | Age: 12 |
| Allow | 对某网络资源的有效的请求行为，不允许则返回405 | Allow: GET, HEAD |
| Cache-Control | 告诉所有的缓存机制是否可以缓存及哪种类型 | Cache-Control: no-cache |
| Content-Encoding | web服务器支持的返回内容压缩编码类型。 | Content-Encoding: gzip |
| Content-Language | 响应体的语言 | Content-Language: en,zh |
| Content-Length | 响应体的长度 | Content-Length: 348 |
| Content-Location | 请求资源可替代的备用的另一地址 | Content-Location: /index.htm |
| Content-MD5 | 返回资源的MD5校验值 | Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ== |
| Content-Range | 在整个返回体中本部分的字节位置 | Content-Range: bytes 21010-47021/47022 |
| Content-Type | 返回内容的MIME类型 | Content-Type: text/html; charset=utf-8 |
| Date | 原始服务器消息发出的时间 | Date: Tue, 15 Nov 2010 08:12:31 GMT |
| ETag | 请求变量的实体标签的当前值 | ETag: “737060cd8c284d8af7ad3082f209582d” |
| Expires | 响应过期的日期和时间 | Expires: Thu, 01 Dec 2010 16:00:00 GMT |
| Last-Modified | 请求资源的最后修改时间 | Last-Modified: Tue, 15 Nov 2010 12:45:26 GMT |
| Location | 用来重定向接收方到非请求URL的位置来完成请求或标识新的资源 | Location: http://www.zcmhi.com/archives/94.html |
| Pragma | 包括实现特定的指令，它可应用到响应链上的任何接收方 | Pragma: no-cache |
| Proxy-Authenticate | 它指出认证方案和可应用到代理的该URL上的参数 | Proxy-Authenticate: Basic |
| refresh | 应用于重定向或一个新的资源被创造，在5秒之后重定向（由网景提出，被大部分浏览器支持） | Refresh: 5; url=  http://www.zcmhi.com/archives/94.html |
| Retry-After | 如果实体暂时不可取，通知客户端在指定时间之后再次尝试 | Retry-After: 120 |
| Server | web服务器软件名称 | Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) |
| Set-Cookie | 设置Http Cookie | Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1 |
| Trailer | 指出头域在分块传输编码的尾部存在 | Trailer: Max-Forwards |
| Transfer-Encoding | 文件传输编码 | Transfer-Encoding:chunked |
| Vary | 告诉下游代理是使用缓存响应还是从原始服务器请求 | Vary: \* |
| Via | 告知代理客户端响应是通过哪里发送的 | Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1) |
| Warning | 警告实体可能存在的问题 | Warning: 199 Miscellaneous warning |
| WWW-Authenticate | 表明客户端请求实体应该使用的授权方案 | WWW-Authenticate: Basic |

## 39、HTTP请求的八种方法

**HTTP协议以及HTTP请求中8种请求方法**

**什么是协议？**

　　协议，是指通信的双方，在通信流程或内容格式上，共同遵守的标准。

**什么是http协议？**

　　http协议，是互联网中最常见的网络通信标准。

**http协议的特点**

　　①通信流程：断开式（无状态）

　　　　　　　　断开式：http协议每次响应完成后，会断开与客户端的连接

　　　　　　　　无状态：由于服务器断开了之前的连接，就无法知晓连接间的关系

　　②内容格式：消息头和消息体

**HTTP请求的方法：**

HTTP/1.1协议中共定义了八种方法（有时也叫“动作”），来表明Request-URL指定的资源不同的操作方式

1、OPTIONS

返回服务器针对特定资源所支持的HTTP请求方法，也可以利用向web服务器发送‘\*’的请求来测试服务器的功能性

2、HEAD

向服务器索与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以再不必传输整个响应内容的情况下，就可以获取包含在响应小消息头中的元信息。

3、GET

向特定的资源发出请求。它本质就是发送一个请求来取得服务器上的某一资源。资源通过一组HTTP头和呈现数据（如HTML文本，或者图片或者视频等）返回给客户端。GET请求中，永远不会包含呈现数据。

4、POST

向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 Loadrunner中对应POST请求函数：web\_submit\_data,web\_submit\_form

5、PUT

向指定资源位置上传其最新内容

6、DELETE

请求服务器删除Request-URL所标识的资源

7、TRACE

回显服务器收到的请求，主要用于测试或诊断

8、CONNECT

HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

注意：

1）方法名称是区分大小写的，当某个请求所针对的资源不支持对应的请求方法的时候，服务器应当返回状态码405（Mothod Not Allowed）；当服务器不认识或者不支持对应的请求方法时，应返回状态码501（Not Implemented）。

2）HTTP服务器至少应该实现GET和HEAD/POST方法，其他方法都是可选的，此外除上述方法，特定的HTTP服务器支持扩展自定义的方法。

**get 和 post区别**

　　区别：

　　　　get请求无消息体，只能携带少量数据

　　　　post请求有消息体，可以携带大量数据

　　携带数据的方式：

　　　　get请求将数据放在url地址中

　　　　 post请求将数据放在消息体中

GET请求请提交的数据放置在HTTP请求协议头中，而POST提交的数据则放在实体数据中；   
GET方式提交的数据最多只能有1024字节，而POST则没有此限制。

## 40、[cookies，session，token](https://www.cnblogs.com/zlj1992/p/7112913.html)

http请求是客户端向服务器发送请求，服务器做出响应，但是有的时候服务器并不知道发出请求的客户端是谁，如何理解这句话呢

如果客户端A通过代理B连接到服务器，这时候A的IP是B分配的，如果这时候连接断了，当客户端再次链接到服务器的时候，IP又变了，这时候服务器以及是不同的客户端，实际上还是同一个客户端

如果多个客户端在同一个局域网通过路由向服务器发送请求，请求的IP均是同一个，这时候服务器以为是同一个的客户端实际上是不同的客户端

因此如何对客户端进行唯一的标识，是需要解决的问题

解决方法：

cookie：

cookie是保存在本地终端的数据，cookie是由服务器生成，发送给浏览器，浏览器以kv的形式进行保存，下一次请求同一个网站的时候会把cookie信息发送给服务器。

cookie的组成有：名称，值，有效域、路径、失效时间、安全标志

通过在http请求中携带的信息进行标识，cookie就是用来解决这个问题的。

服务器端返回给客户端一个请求响应的时候，会包含set-cookies，意思是指示客户端建立cookie，并且在后续的http请求中发送这个cookie直到这个cookie过期。

一个cookie的设置以及发送过程分为以下四步：

客户端发送一个http请求到服务器端

服务器端发送一个http响应到客户端，其中包含Set-Cookie头部

客户端发送一个http请求到服务器端，其中包含Cookie头部

服务器端发送一个http响应到客户端

session：

session的意思是“会话”，客户端与服务器的一次请求就产生一个session，服务器使用session将用户的信息临时保存在服务器上，用户离开网站session就会销毁。这种方式相对于cookie而言比较安全，但是也会存在问题，如果服务器做了负载均衡的话，那么下一次操作到别的服务器，session就会丢失。

token：

token是“令牌”是用户身份的验证方式，包含：uid，time，sign

**cookie 和session的区别**

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
   考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能  
   考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：  
   将登陆信息等重要信息存放为SESSION  
   其他信息如果需要保留，可以放在COOKIE中

**token 和session 的区别**

  Session 是一种HTTP存储机制，目的是为无状态的HTTP提供的持久机制。所谓Session 认证只是简单的把User 信息存储到Session 里，因为SID 的不可预测性，暂且认为是安全的。这是一种认证手段。 而Token ，提供的是 认证 和 授权 ，认证是针对用户，授权是针对App 。其目的是让 某App有权利访问 某用户 的信息。这里的 Token是唯一的。不可以转移到其它 App上，也不可以转到其它 用户 上。 转过来说Session 。Session只提供一种简单的认证，即有此 SID，即认为有此 User的全部权利。是需要严格保密的，这个数据应该只保存在站方，不应该共享给其它网站或者第三方App。 所以简单来说，如果你的用户数据可能需要和第三方共享，或者允许第三方调用 API 接口，用 Token 。如果永远只是自己的网站，自己的 App，用什么就无所谓了。

  token就是令牌，比如你授权（登录）一个程序时，他就是个依据，判断你是否已经授权该软件；cookie就是写在客户端的一个txt文件，里面包括你登录信息之类的，这样你下次在登录某个网站，就会自动调用cookie自动登录用户名；session和cookie差不多，只是session是写在服务器端的文件，也需要在客户端写入cookie文件，但是文件里是你的浏览器编号.Session的状态是存储在服务器端，客户端只有session id；而Token的状态是存储在客户端。